코딩과 장벽

이재현

2022.11.11

Abstract

사람들은 일상적으로 코딩과 비슷한 활동들을 하고 있다. 그러나 자신이 구현하고자 하는 일을 프로그래밍 언어 체계에 맞춰서 기술해야 한다는 것은 컴퓨터 전공자와 비전공자를 구분짓는 하나의 장벽이다. Copilot의 인공지능이 제공하는 코드 추천 서비스로 그 장벽을 넘을 수 있을 것도 같지만, 합성된 프로그램이 정확한지를 검증할 수 없기에 아직 비전공자들이 사용하기에는 부족하다. 전통적인 프로그램 합성 기법이 제공하는 정확성을 Copilot과 같은 인공지능 기반 합성 기법에 접목시킬 수 있다면, 그 장벽을 무너뜨리는 것이 가능할 지도 모른다.

공학자가 할 수 있는 멋진 일들 중 하나는 고급 기술에 대한 접근 장벽을 허무는 일이라고 생각한다. 에디슨과 테슬라를 비롯한 전기공학자들은 복잡한 전자기학과 일상 사이의 장벽을 허물었다. 그 덕분에 우리는 전자(electron)가 무엇인지 모르더라도 전자기기를 사용하는 데 아무 제약을 받지 않는다. 코딩에 관한 첫 장벽은 이진법으로 기술된 컴퓨터의 명령어들과 개발자의 논리 체계 사이의 장벽이었을 것이다. 그 첫 번째 장벽을 허문 공학자 중 한명은 그레이스 호퍼이다. 그녀는 기계어와 프로그래밍 언어를 연결하는 다리인, A-0라는 최초의 컴파일러를 만들었다. 이 컴파일러를 통해 개발자에게 친숙한 프로그래밍 언어를 기계어로 번역할 수 있게 되었다. 그녀 덕분에 지금의 개발자들은 Python, C, OCaml 등의 다양한 프로그래밍 언어들로 자신의 논리를 표현한 프로그램을 작성할 수 있다.

그렇다면 코딩에 대한 두 번째 장벽은 무엇일까? 그것은 컴퓨터 전공자와 비전공자를 구분짓는 장벽이라고 생각한다. 사람들은 일상적으로 저마다의 방법대로 코딩을 하고 있다. 예를 들어, 어머니께서는 요리를 하실 때 어머니만의 규칙을 가지고 계신다. 육수를 우리는 동안에는 채소를 손질해야 하며, 육수가 다 우러났다면 미리 준비해 둔 양념장을 꺼내신다. 이러한 과정은 프로그래밍 언어의 절차문과 반복문을 연상하게 한다. 너무나 일상적인 일이지만, 어머니께서 규칙을 정하셨고, 그 규칙을 이행하셨다는 점에서 정해진 프로그래밍 언어 문법을 그 의미 구조(semantics)에 따라 이행하는 프로그래밍 언어의 동작 원리와 비슷하다. 이처럼 우리의 삶 속에 규칙과 논리가 익숙하게 자리 잡았음에도 불구하고, 모두가 코딩을 하지는 않는다. 코딩은 컴퓨터 전공자들만 할 수 있는 일로 여겨지고는 한다. 비전공자들이 코딩을 접하는 것을 꺼려하는 이유 중 하나는, 자신이 무엇을 하고 싶은지를 프로그래밍 언어라는 정해진 형태로 표현하는 것에서 어려움을 느끼기 때문이다. 아버지께서는 학생들을 가르치는 일을 하신다. 학생들의 숫자가 너무 많아서, 중간고사 답안지를 채점하실 때 자동으로 채점을 해 주는 프로그램이 있었으면 좋을 것 같다고 이야기하셨다. 채점하는 논리 자체는 특정 정답 문자열이 답안 속에 포함되어 있는지를 검사하는 간단한 논리이다. 하지만, 이것을 프로그래밍 언어라는 체계로 표현해야 한다는 점에서 장벽을 느끼셨기에 채점 프로그램을 실현하지 못하셨다.

최근 Github의 Copilot이라는 도구의 등장으로, 누구나 쉽게 코딩할 수 있는 미래가 보이는 듯 하다. 일례로 Copilot은, 아버지께서 "답안에 '수요와 공급'이라는 문자열이 들어가 있다면 점수에 10점을 더해 줘" 라는 프로그램의 명세(specification)을 제공하시면 그에 맞는 프로그램 코드를 제시하는 방식으로 동작한다. 이것은 Github에 저장된 방대한 코드 데이터베이스를 학습한 인공지능 모델을 통해 가능한 일이다. 다시 말해, Copilot은 지금까지 축적된 프로그램 예시들을 바탕으로, 프로그램의 명세가 주어지면 그에 따라 작동하는 그럴듯한 프로그램을 생성하여 제시하는 도구이다. Copilot은 기존 개발자들에게 편리성을 더해 주는 도구로 주목을 받고 있다. Copilot을 통해서 반복되는 코드를 써야 하는 번거로움을 줄일 수 있다. 그러나 비전공자들에게 Copilot은 여전히 사용하기 까다롭고 어려운 도구라고 생각한다. Copilot이 그럴듯한, 즉 틀릴 수도 있는 프로그램을 제시한다는 점에서 그렇다.

비전공자들이 Copilot을 활용하기 가장 어려운 이유는, Copilot이 정확성을 보장할 수 없는, 그럴듯한 프로그램을 제시한다는 점이다. Copilot 인공지능은 Github에 올라온 틀린 코드들도 다른 코드들과 함께 학습한다. 그렇기에 Copilot에 의해 합성된 프로그램들이 항상 의도대로 동작한다는 것을 보장할 수는 없는 일이다. 아버지께서 자동 채점 프로그램을 합성하셨다면, 거기에는 오류가 숨어 있을 확률이 있다. 그 오류는 예외적이어서

무시할 만한 것일 수도 있지만, 학생들의 시험 점수에 영향을 미치는 치명적인 오류일 가능성도 배제할 수 없다. 그리고 운 좋게 그 프로그램이 잘못되었다는 것을 인지했다 하더라도, 오류의 원인을 진단하고 수정하는 것은 아직 비전공자가 할 수 있는 영역이 아니다. 왜냐하면, 코드에서 오류를 찾고 고치는 일에는 프로그래밍 언어에 대한 이해가 필수적이기 때문이다. 이처럼 비전공자가 Copilot과 같은 인공지능 합성기를 완전히 신뢰하고 사용할 수는 없는 것이 현실이다. 따라서, 코딩에 대한 두 번째 장벽을 완전히 허물기 위해서는 정확성을 보장할 수 있는 프로그램 합성기를 만들어야 한다.

Copilot처럼 상업적으로 유명한 프로그램 합성기가 나오기 이전에도 프로그램 합성은 꾸준히 연구되고 있었다. 연구자들은 논리와 탐색(search) 기반으로 사용자의 명세에 맞는 프로그램을 합성하고자 했다. 가장 간단한 프로그램에서부터 시작해서 프로그램을 점점 확장해 나가면서 명세에 맞는 프로그램을 탐색하는 방식이다. 비록 확장성이 떨어지고, 때로는 느리지만, 합성된 프로그램의 정확성을 검증하고 보장할 수 있다는 장점이 있다. 인공지능 바탕의 합성기는 인공신경망을 활용하기 때문에 프로그램이 어떻게, 왜 합성되었는지를 설명하기 어렵다. 하지만 전통적인 방법으로 합성된 프로그램은 설명 가능한 알고리즘을 바탕으로 만들어졌다. 그렇기에 합성된 프로그램이 정말 사용자의 의도대로 작동하는지, 만약 그렇지 않다면 어떠한 합성 과정이 문제가 되었는지를 진단 가능하다.

그렇다면, 전통적인 프로그램 합성과 인공지능 기반의 프로그램 합성의 장점만을 취해서 설명 가능하면서 확장성 있는 합성기를 만들 수도 있을 것이다. 한 가지 방법은, 전통적인 알고리즘으로 프로그램을 합성하되, 그 과정에서 인공지능의 지도를 받는 것이다. 인공지능이 그럴듯한 프로그램들을 추천하고, 합성기가 그 중에서 정확한 프로그램을 골라 내는 방식이다. 그러면 프로그램 탐색 공간을 효과적으로 줄일 수 있기 때문에 전체 합성에 걸리는 시간을 줄일 수 있다. 또 다른 방법으로는, 인공지능을 통해 사용자의 명세를 구체화하고, 그것을 전통적인 프로그램 합성기에 넘기는 방식이다. 이 방법으로는 합성기의 범용성을 확보하고 정확성을 어느 정도 보장할 수 있을 것이다.

컴퓨터 전공자와 비전공자를 구분짓는, 정해진 프로그래밍 언어에 의한 코딩이라는 장벽을 무너뜨리는 것은 꿈만 같은 일이다. 그 장벽은 너무나도 높아 보이지만, Copilot이 그 장벽을 허물 수 있는 가능성을 보여주고 있기도 하다. 그 장벽이 정말 무너진다면, 누구나 일상적으로 코드를 작성해서 반복적인 작업들을 컴퓨터에게 맡길 수 있을 것이다. 결국 사람들에게 가장 소중한, 시간을 벌어다 주는 위대한 혁신이 될 것이라 생각한다.